2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Generation of UVM compliant Test Benches for Automotive Systems using IP-XACT with UVM-SystemC and SystemC AMS*

Ronan Lucas, Magillem Design Services, Paris, France, lucas@magillem.com

Philippe Cuenot, Continental Automotive France, Toulouse, France, philippe.cuenot@continental-corporation.com

Marie-Minerve Louërat, Yao Li, Zhi Wang, Jean-Paul Chaput, François Pêcheux, Ramy Iskander, LIP6, UMR 7606 SU-UPMC/CNRS, Paris, France, marie-minerve.louerat@lip6.fr

Martin Barnasconi, NXP Semiconductors, Eindhoven, Netherlands, martin.barnasconi@nxp.com

Thilo Vörtler, Fraunhofer IIS, Design Automation Division, EAS, Dresden, Germany, thilo.voertler@eas.iis.fraunhofer.de

***Abstract*—This paper will present a methodology and flow to automate the test bench creation for automotive heterogeneous HW/SW systems, using SystemC, SystemC-AMS and IP-XACT. The UVM foundation elements such as test, environment, UVC (Universal Verification Component), transactions and associated configuration objects are introduced, which are packaged by means of IP-XACT vendor extensions. The benefit is to facilitate the (re)use of UVM components and environments by providing a readable and configurable test platform description, to trace the requirements of tests, and to generate automatically the entire UVM environment and simulation build flow after configuration of the test scenario. The automation technology is based on IP-XACT and uses new capabilities of the Magillem tools solution.**

Keywords— Simulation, Verification, EDA Automation, SystemC, SystemC-AMS, IP-XACT, UVM

I. INTRODUCTION

The complexity of electronic systems for the automotive industry is increasing. These systems are characterized by having close interaction between software (SW), hardware (HW) and analog components. The Electronic Control Unit (ECU) is a heterogeneous system, since it contains digital, analog low-voltage and high-voltage electronics, combined with software running on an embedded processor. Furthermore, automotive systems are safety critical systems, and as a consequence, verification of all related requirements is mandatory.

Therefore, to design these safety-critical systems, the engineers more often require a virtual prototype (VP) of the hardware architecture. Using the VP, software engineers are able to debug the HW/SW automotive system before the actual availability of the hardware prototype. Later on, when the HW is available, in the form of an FPGA or test chip, a strong benefit is to reuse the test cases and test benches used for the VP, and to apply them as well to validate the hardware prototype.

To introduce this way-of-working with a stronger focus on reuse, we propose to generate test benches following the Universal Verification Methodology (UVM) principles [1] using IP-XACT [2], which are used to bridge the gap between verification and validation. The proposed techniques are applied on a smart power supply module, which is part of an automotive Electronic Control Unit, to demonstrate the automation capabilities for verification methods.

The paper is organized as follows. Section II describes the related work in the areas of methodologies and verification tools. Section III introduces the verification methodology and framework using UVM in SystemC. In Section IV, the IP-XACT extensions are presented for UVM-SystemC. The automotive use case is presented in Section V. Section VI shows the automation techniques based on IP-XACT, supported in the Magillem design environment. Then, in Section VII, the experimental results are presented based on the automotive application. Finally, Section VIII gives the conclusions and future directions.

*Sponsored by the European Commission within the 7th Framework Program for Research and Technological Development (FP7/ICT 287562)

II. STATE OF THE ART

Driven by the complexity of embedded systems and the safety critical HW/SW systems, different verification methodologies for digital systems have been developed and promoted by EDA vendors over the last decade, evolving towards the UVM as part of an industry supported standardization effort in Accellera [1]. The UVM standard has arisen from several attempts to create structured and reusable verification environment for digital systems as shown in Figure 1. It finds its origin in quite a number of former verification technologies and methodologies (e.g. ***e***RM, VRM, AVM, VMM, OVM) [3-6]. The development of the UVM standard includes an open-source class library to create verification components, to support design and verification engineers in the creation of digital test benches.

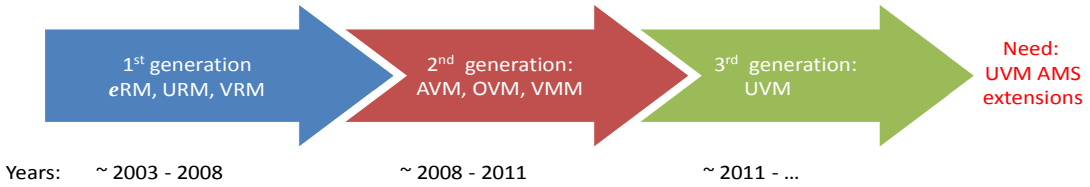


Figure 1: Historical perspective on verification Methodologies

Till recently, dedicated verification languages were used for verification, such as vera, ***e***, or SystemVerilog. The use of the SystemC language to enable more abstract system-level verification is not yet supported. Therefore a UVM standard compliant language definition and reference implementation in SystemC/C++ [7] has been developed by NXP in the context of the VERDI project [8]. Moreover, the AMS extensions for SystemC [9] are also used to handle the verification of mixed-signal systems, which can be applied within such UVM environment.

III. VERIFICATION FRAMEWORK UVM-SYSTEMC

UVM for SystemC defines all the essential features to create a UVM standard compliant verification environment. It contains many built-in capabilities dedicated to verification, such as test and test bench creation, configuration, phasing, comparing, scoreboarding and reporting [10].

The main principle of UVM is to help the verification engineer to describe abstract test scenario in terms of a stream of transactions. These transactions are further refined into signals and sent to the Device Under Test (DUT) by a driver. The monitor collects the output signals from the DUT and stores them. From a relevant set of output signals, the analysis utility functions compute the performances of the DUT and send them to the scoreboard. The scoreboard compares the resulting performance of the DUT with the reference ones.

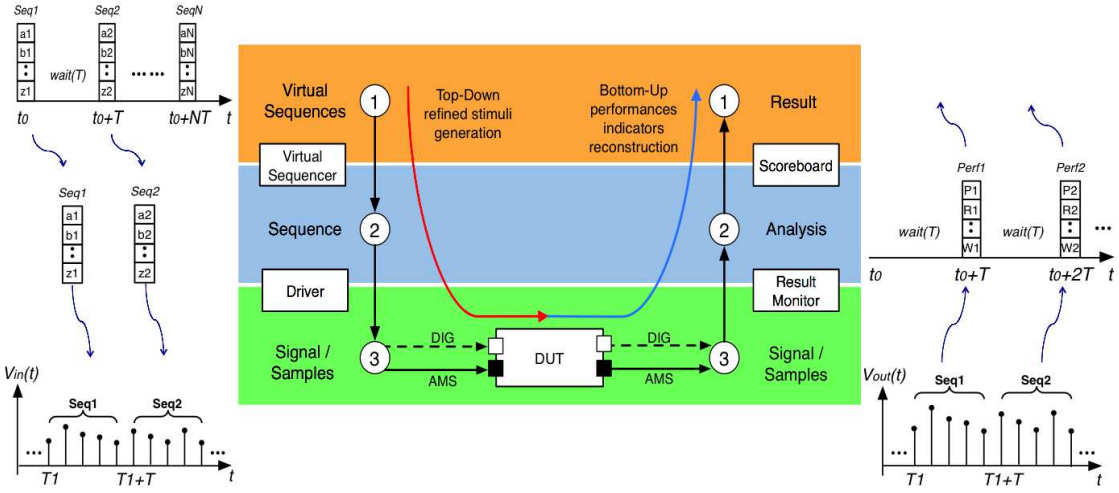
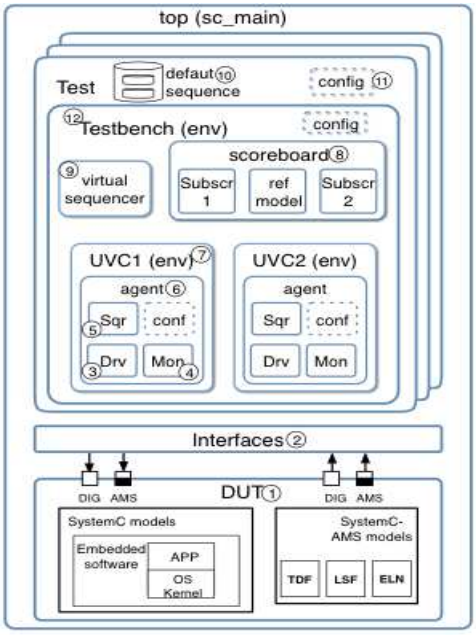


Figure 2: UVM based layering and the signal/result synchronization

Figure 2 illustrates the top-down decomposition of test sequence stimuli in UVM, as well as the bottom-up reconstruction of performance indicators for verification. The sequencers/drivers wait for a period **T**. The

monitoring thread waits for the same period, so the emission of stimuli and the reconstruction of performance indicators are kept synchronous during the simulation.

In order to enable horizontal or vertical reuse, a UVM environment is built in a structured way with specific communication interfaces between its components. The typical verification environment architecture is illustrated in Figure 3. It consists of multiple UVM components (7). The corner stone of the UVM hierarchy is the agent component (6). An agent component instantiates all the components that are necessary to drive (respectively monitor) the signals to (respectively from) the DUT.



DUT : Design Under Test
APP : Software application executed on DUT
TDF : Timed Data Flow signal
LSF : Linear Signal Flow
ELN : Electrical Linear Network
DIG : Digital physical signal
AMS : Convertor port between digital and analog signal
UVC : Universal Verification Component (uvm_env)
Agent (uvm_agent)
Sqr: Sequencer (uvm_sequencer)
Drv: Driver (uvm_driver)
Mon: Monitor (uvm_monitor)
Scoreboard: (uvm_scoreboard)
Subscr: Subscriber (uvm_subscriber)
Test bench (uvm_env)
Test (uvm_test)

Figure 3: UVM test environment and terminology

Typically, the agent instantiates three components: a sequencer (5), a driver (3) and a monitor (4). The agent components use a TLM interface with the higher levels in the test bench architecture and use a physical level interface (2) to communicate with the DUT. The DUT (1) is composed of digital and AMS parts, some of them eventually executing software. The drivers translate the sequence of data transaction to SystemC discrete event signals for the digital IPs and to SystemC-AMS TDF (Timed Data Flow) samples for AMS IPS and send to DUT port (Fig. 2 and Fig. 3).

The test bench (12) is defined as the complete verification environment which instantiates and configures (11) the universal verification components (7), scoreboard (8) and virtual sequencer (9). The virtual sequencer controls all sequencers at top level. The sequencers handle all transactions and send them to the drivers. Sequences (10) encapsulate sequence items, defined as transactions. The monitor (4) receives and accumulates the output signal level vectors/samples. Utility functions in monitors collect signals and the analysis functions compute system performances. The scoreboard (8) performs end-to-end checking by comparing the golden model reference performance specifications with the extracted performances.

IV. IP-XACT EXTENSIONS FOR UVM-SYSTEMC

The IP-XACT metadata is used to provide a unified specification for a verification component to exchange and share compatible components from multiple companies or services. More specifically, the creation of UVM extensions in IP-XACT enable efficient assembly and configuration of test bench, test and top level elements by generating the relevant SystemC and SystemC-AMS views necessary to conduct verification.

The UVM architecture is structured in layers from the top level to the virtual sequencers, and UVCs, which can be reused independently of each other. To maintain reusability, the IP-XACT description is following the same hierarchical structure using the concept of design, component and hierarchical view defined in the IEEE1685 standard schema.

In order to identify each UVM specific component, IP-XACT vendor extensions are introduced in the component definition of the schema. This extension defines the class from which the component is derived from (cf. Figure 3).

The instantiation of the different UVCs in the IP-XACT design and the interconnections with the scoreboard will assemble the test-bench. The UVCs and scoreboard are hierarchical components with TLM interfaces. These hierarchical interconnections allow connecting accurately subscribers and monitors.

The top level instantiates the DUT, the test and (virtual) interfaces used to connect the UVCs to the DUT by using the UVM configuration database mechanism. The IP-XACT representation of a (virtual) interface is a SystemC or SystemC-AMS component extended with an *interface* attribute and contains a physical interface and a logical interface. The physical interface is a list of SystemC or/and SystemC-AMS signals definition that will be encapsulated in the interface and used to connect the DUT. It is represented by the IP-XACT ports and can be eventually regrouped in a bus interface definition (e.g. AHB, OCP, ...). The logical interface is described as a bus interface and represents the interconnection with the UVC through the UVM configuration mechanism. The bus definition of the bus interface of the UVC and virtual interface must be the same. The IP-XACT abstraction definition and port map descriptions are not defined in this bus interface to avoid any actual connections.

The UVM configuration database is a central resource database to store and retrieve any type specific information of UVM and non-UVM objects at any place in the verification environment. IP-XACT parameters are used to specify if an element is stored in the configuration database. UVCs and interface components can be enabled or not in this database depending on a boolean parameter. Properties and specific variables settings are also supported by the configuration database. For example, the property whether a UVC is active or passive is specified in a parameter. The switches to enable checking and coverage collection in the monitors depend on boolean parameters. All these parameters are user-defined, specified in a view section relative to the configuration and is introduced using the IP-XACT vendor extension attribute *uvmCfgDatabase*.

Moreover, the configuration of the test bench and verification components can be performed with the help of configuration objects. From the test bench configuration, the user can configure the different UVCs through nested configuration objects associated to each UVC. In turn, the configuration objects contain handles for their sub-component configuration objects. The prototype of each configuration object must be defined within extensions in the IP-XACT component description. A configuration is defined by a name and an unbounded list of parameters and/or nested configuration description. The parameters are user-defined and specified in the view referenced in the configuration description. The resolution is “user” indicating that the value is specified by the user input and the new value is stored in a design or design configuration description.

V. USE CASE DESCRIPTION

The selected use case as DUT to evaluate the technology is a subset of a smart power supply module extracted from an automotive Electronic Control Unit (ECU) of an engine management system developed by Continental Automotive France. It is integrated in an ASIC including a set of voltage regulator blocks providing several supply ranges: an internal and external reset circuitry for the complete ECU, and an SPI communication interface. Additionally, a complex state machine for safety monitoring purpose is integrated in the ASIC supervising the ECU via the same SPI interface with a specific protocol. The prototype built for the definition of concept for the automation of the UVM verification methodology is representative of the ASIC, but scaled down for tutorial and experimentation purpose. It is implemented using SystemC and SystemC-AMS class libraries and composed with IP-XACT.

According to the UVM methodology, the DUT verification will be performed by simulation. The test bench is implemented with the UVM-SystemC class libraries. The UVCs are defined according to the DUT subset behavior, in order to be reused across several design verification projects. Moreover, the UVCs are configured via the UVM configuration object to facilitate reuse, and in particular to allow driver configuration for variable stimuli definition. The stimuli itself are captured in a text file defining the transactions to drive the sequence of the respective UVC sequencer. These stimuli files are designed using a specific format to overwrite the driver transaction and are shared with the tool environment for tape-out silicon laboratory validation using Labview [11] configurable scripts.

The DUT is organized with four main functionalities: a single analog voltage regulator captured in Time Data Flow (TDF) domain of SystemC-AMS, a logical controller of the voltage regulator similar to ASIC reset control, a mixed function supervising the voltage value of the regulator and building frame for communication mimicking the ASIC specific protocol, and finally a digital SPI interface defined at transactional level identical to the one used in the ASIC. The UVM test-bench is organized with 4 UVCs controlled by a virtual sequencer, as depicted in the architecture view in Figure 4 below.

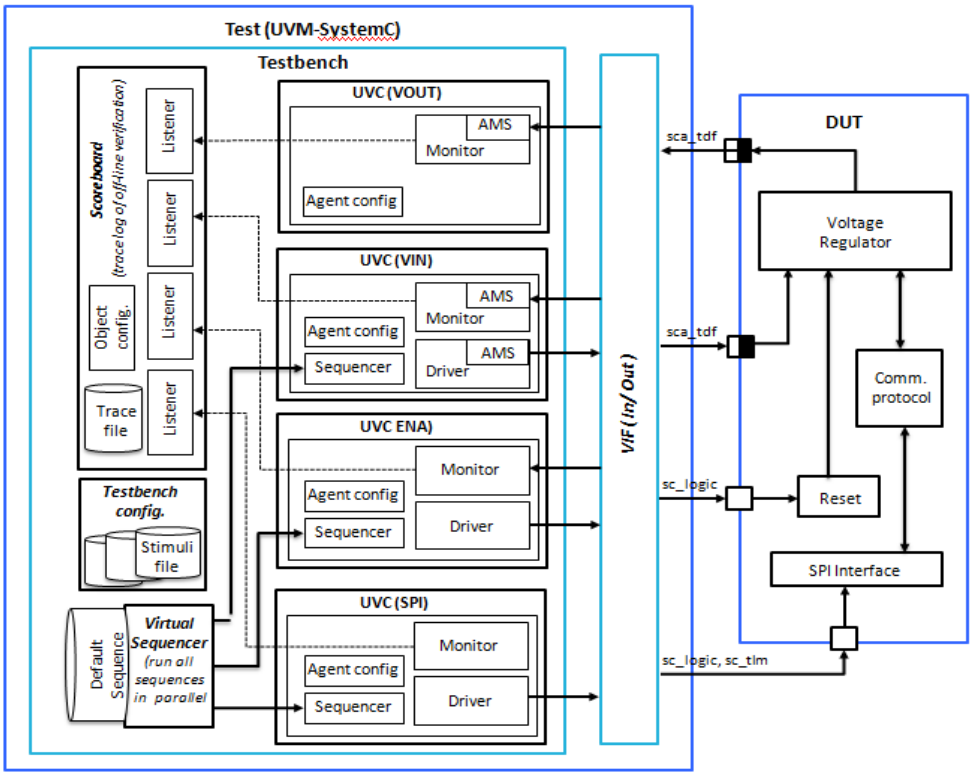


Figure 4: UVM-SystemC verification environment for a power supply module

The first AMS-UVC agent **(VIN)** is driving a TDF input signal of the DUT power voltage regulation. A configuration object is associated to this agent to define the characteristics necessary for the agent simulation as a point to the virtual interface, the active or passive state of the agent, the reference to the stimuli file, the configuration for interpolation between stimuli single value, and the AMS configuration for simulation time step. The configuration object is derived from *uvm_object*. The UVC itself is derived from class *uvm_env* and is part of the nested hierarchy of the test bench component derived from *uvm_env*. The configuration is defined in the test layer (derived from *uvm_test*) and instantiates a test bench and configuration object. The driver of VIN, derived from *uvm_driver*, is feed by transactions of type *uvm_sequence* from the sequencer. The transaction (*uvm_sequence_item*) delivers a vector containing *tdf_value* and *time_stamp* pairs to the VIN driver.

To synchronize the AMS domain of the DUT with the driver, an AMS (TDF) module is instantiated in the driver, which is natively synchronized using a SystemC-AMS converter port between the TDF and discrete event domain. The AMS simulation time step of this component is scheduled by the value of the UVC configuration object, propagated through the UVM component hierarchy via a class attribute. The AMS module embeds the TDF port interface in the driver to pass the analog value and time stamp, where interpolated values are managed in the AMS component. In the driver the virtual interface is bound to the output port of the AMS component. The driver stimuli are preprocessed from the stimuli text file, and recovered in the body method of the sequence to generate the transaction. The VIN monitor is using the same transactions as the driver, captured in a one dimensional vector. In a similar way, the monitor instances contain an AMS TDF module to ensure the synchronization with the AMS TDF domain, and bind the virtual interface to input port of this TDF module.

A second logical UVC agent **(ENA)** is setting or resetting the digital reset input of the DUT via the control of the virtual interface. This UVC is also associated with a configuration object defining the configuration of the agent as a pointer to the virtual interface, the active or passive state of the agent, and the reference to the stimuli file. Similar to the above VIN agent, its driver is fed by the value of a transaction as vector of *logic_value-time_stamp* pairs. Identically, the sequence is preprocessing the transaction by reading the values from the text file recovered from label definition in the data base. The monitor is capturing the same transaction to be written in the listener of the scoreboard.

The third logical agent **(SPI)** is managing the transaction for SPI communication by control of the virtual interface. It has also a configuration object similar to one of the ENA agent, but in addition it includes a pointer to another SPI configuration object for defining timing characteristics of the SPI communication. The driver, sequence and monitor are similar to control in the ENA agent; the only change is that the driver is controlling the SPI timing configuration thanks to the configuration object accessed thought the component hierarchy and object configuration.

The fourth AMS UVC agent **(VOUT)** is a passive component that only monitors the output of the DUT. It has a configuration object necessary to configure the simulation time step for the AMS part and a pointer to the virtual interface. The construction for the monitor is identical to the one in the VIN agent.

The UVCs are strongly coupled due to the dependencies in the DUT. The UVC sequences are executed in parallel, started by the virtual sequence and have a common reference time t0. The scoreboard collects all recorded transaction of the UVCs via the listeners to build a log file, storing data and time stamp for an off-line post-verification. The scoreboard has a configuration object to configure the filename for the log file, and uses start-time and stop-time of the recording to ensure similar recording conditions to the laboratory environment for off-line comparison (limitation capacities constraints). The object configurations are nested within the hierarchy and passed-through via the UVM configuration database, which is declared at the top-level of the test to facilitate configuration and reuse.

The challenge for the automation is to be able to (re)use agents and scoreboard components as structural organization and to generate all the glue code necessary for the test (*uvm_test*) and test bench (*uvm_env*) generation, including the initialization and nesting of the object configuration passed through the hierarchy of the database. The dynamic aspects of using a virtual sequencer processing sequences for dynamic behavior is partly written, as the complete sequence stimuli are defined from the configuration issued from the stimuli text file.

VI. EDA AUTOMATION USING IP-XACT

Magillem tooling [12] is able to generate the whole UVM-SystemC test environment (in .h and .cpp files) for top-level, test, test bench and also test bench configuration and virtual sequencer, based on the hierarchical IP-XACT description of the platform.

The meta-data are loaded in a dedicated data model during a preliminary step. From this structured model we elaborate the assembly of each design depending on the interconnection and the configuration of the instances. Once these pre-required steps are realized, the UVM generator can be executed to provide the different output files. For each generated file, a specific sub-generator is associated, ensuring modularity in the architecture of the tool. Additionally, a template mechanism, one per generator, is introduced to provide flexibility and takes into account some coding style and text formatting. In these templates, some specific tags started and ended by the '@' character are inserted between comments or SystemC code and will be replaced by data during the generation.

Listing 1 shows an example of the generator. The left column shows the template used. The right column gives the generated code. The engine of each sub-generator use the IP-XACT IEEE1685 standard API (Application Programming Interface), the Tight Generator Interface, to query the model and get the meta-data used to compute the final data that will be dumped in the template.

```
Int sc_main(int, char*[]) {

//instantiate the dut
@SCINSTANCES@

//instantiate the virtual
interfaces
@VIRTUALINTERFACES@

// register virtual interface
via configuration mechanism
@REGISTERVIRTUALINTERFACE@

// Connect the Virtual
Interfaces to the dut
@CONNECTIVITY@
```

```
int sc_main(int, char*[]) {

// instantiate the DUT
V_regulator_test* dut = new V_regulator_test("dut");

// instantiate the UVM agent virtual interface
v_regulator_if_ena* vif_v_regulator_ena = new v_regulator_if_ena();
v_regulator_if_vin* vif_v_regulator_vin = new v_regulator_if_vin();
v_regulator_if_spi* vif_v_regulator_spi = new v_regulator_if_spi();
v_regulator_if_vout* vif_v_regulator_vout = new v_regulator_if_vout();

// register virtual interface via configuration mechanism
uvm_config_db<v_regulator_if_ena*>::set(0, "*", "vif_v_regulator_ena",
vif_v_regulator_ena);
uvm_config_db<v_regulator_if_vin*>::set(0, "*", "vif_v_regulator_vin",
vif_v_regulator_vin);
uvm_config_db<v_regulator_if_spi*>::set(0, "*", "vif_v_regulator_spi",
vif_v_regulator_spi);
uvm_config_db<v_regulator_if_vout*>::set(0, "*", "vif_v_regulator_vout",
vif_v_regulator_vout);

// Connect the Virtual Interface
dut->Vin_port(vif_v_regulator_vin->sig_Vin);
dut->Enable_V(vif_v_regulator_ena->sig_Enable_V);

dut->Trans_spi_cs_port(vif_v_regulator_spi->sig_Tspi_CS_V);
dut->Trans_spi_control_port(vif_v_regulator_spi->sig_Tspi_CTRL_V);
dut->Trans_spi_sdi_port(vif_v_regulator_spi->sig_Tspi_SDI_V);
dut->Trans_spi_sdo_port(vif_v_regulator_spi->sig_Tspi_SDO_V);
```

Listing 1: Template and generated code

VII. EXPERIMENTAL RESULTS

The IP-XACT-based verification methodology aims at facilitating the use of UVM objects by providing a simple, configurable, and readable description of a UVM verification component (UVC), to automate the creation of the test environment, and with this, to increase the test writer efficiency by letting him/her focus on sequences and tests. From a verification point of view, the user builds the UVM platform by selecting the UVCs from an IP-XACT library, and instantiate, configure and interconnect these to build the test bench. Test and top-level are similarly assembled and configured.

As shown in Figure 5, the Magillem Platform Assembly tool offers a graphical or TCL scripting interface to compose the IP-XACT platform and generates the different UVM layers (top-level, test and test bench) in the SystemC and SystemC-AMS language.

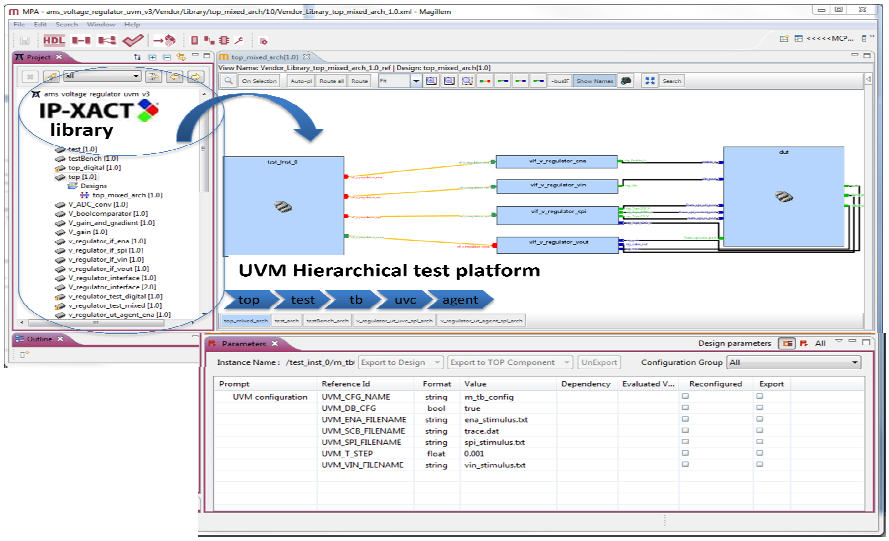


Figure 5: Magillem Platform Assembly tool to build the UVM test environment

A parameter dedicated view facilitates the configuration of the platform of test and propagates through the hierarchy the parameter values to the sub-element using the IP-XACT description of the different configuration object associated to their IP-XACT component.

Once the generation is completed, the following files have been generated:

- *sc_main.cpp* : instantiating the DUT, virtual interfaces and test and connecting the DUT and virtual interfaces
- *test.h* and *test.cpp*: containing the declaration and the implementation of the configuration object, the build phase and the run phase.
- *testbench.h* and *testbench.cpp* : containing the declaration and instantiation of the UVCs, the implementation of build phase and the connect phase
- *testbench_config.h*: the top level configuration object containing the declaration of each sub-configuration object and the list of all global parameters
- *virtual_sequence.h*: referring all the sequencers in the platform that will be used to create the sequence items and send them to the driver.

VIII. CONCLUSIONS

This paper demonstrated test and test bench automation, by means of generating the entire verification environment in UVM-SystemC and SystemC-AMS, thanks to the use of the UVM standard and extended IP-XACT descriptions. The approach has been demonstrated on a voltage regulator ASIC as part of a smart power supply module, extracted from an automotive Electronic Control Unit (ECU) of an engine management system. It allows to facilitate the ASIC verification, by reusing existing test components and defining associated configuration, to complete the verification phase in automated process.

The automation can be extended by the new traceability concept available in Magillem to permit the execution by a requirement driven verification using tracing down to the test component configuration. Moreover, the stimuli text file driving the test sequence enables us to reuse the test scenario definition between the verification and validation phase. These features under experimentation on the same use case will complete the demonstration of IP-XACT extension capabilities supported by a verification and validation methodology.

ACKNOWLEDGMENT

This work was funded by the project Verification For Heterogeneous Reliable Design and Integration (VERDI), which is supported by the European Commission within the 7th Framework Program for Research and Technological Development (FP7/ICT 287562).

REFERENCES

[1] Accellera Systems Initiative, Universal Verification Methodology (UVM) standard, http://www.accellera.org/downloads/standards/uvm/

[2] Accellera Systems Initiative, IP-XACT standard, http://www.accellera.org/activities/committees/ip-xact/ and IEEE Std. 1685-2009, http://standards.ieee.org/getieee/1685/download/1685-2009.pdf

[3] A. Koczor, W. Sakowski, “SystemC library supporting OVM compliant verification methodology”, IP Embedded System Conference and Exhibition (IP-SoC), December 2011

[4] M. F. S. Oliveira, C. Kluznik, W. Mueller, W. Ecker, V. Esen, “A SystemC Library for Advanced TLM Verfication”, Design and Verification Conference & Exhibition (DVCon), February 2012

[5] A. Sarkar, “Verfication in the trenches : A SystemC implementation of VMM1.2”, http://www.vmmcentral.org/vmartialarts/2010/12/verification-in-the-trenches-a-systemc-implementation-of-vmm1-2/

[6] IEEE Standard Association, “1647-2011 - IEEE Standard for the Functional Verification Language e “, http://standards.ieee.org/findstds/standard/1647-2011.html

[7] IEEE Computer Society, 1666-2011 IEEE Standard SystemC Language Reference Manual, http://standards.ieee.org/findstds/standard/1666-2011.html

[8] Y. Li, Zh. Wang, M. M. Louërat, F. Pêcheux, R. Iskander, Ph. Cuenot, M. Barnasconi, Th. Vörtler, K. Einwich : “Virtual Prototyping, Verification and Validation Framework for Automotive Using SystemC, SystemC-AMS and SystemC-UVM”, Embedded Real Time Software and Systems (ERTS2), February 2014, Toulouse

[9] Accelera Systems Initiative, SystemC AMS 2.0 Standard, http://www.accellera.org/downloads/standards/systemc/

[10] M. Barnasconi, F. Pêcheux, T. Vörtler, “Advancing system-level verification using UVM in SystemC”, Design and Verification Conference & Exhibition (DVCon), Mar. 2014, USA

[11] LabVIEW System Design Software, http://www.ni.com/labview/

[12] Magillem Platform Assembly, http://www.magillem.com/eda/